# MNIST with Docker

Siddhant Patny (sp5331)

Instructions:

- Steps: Create a Vagrant File with Docker Preinstalled
- Run MNIST inside Docker on the virtual box

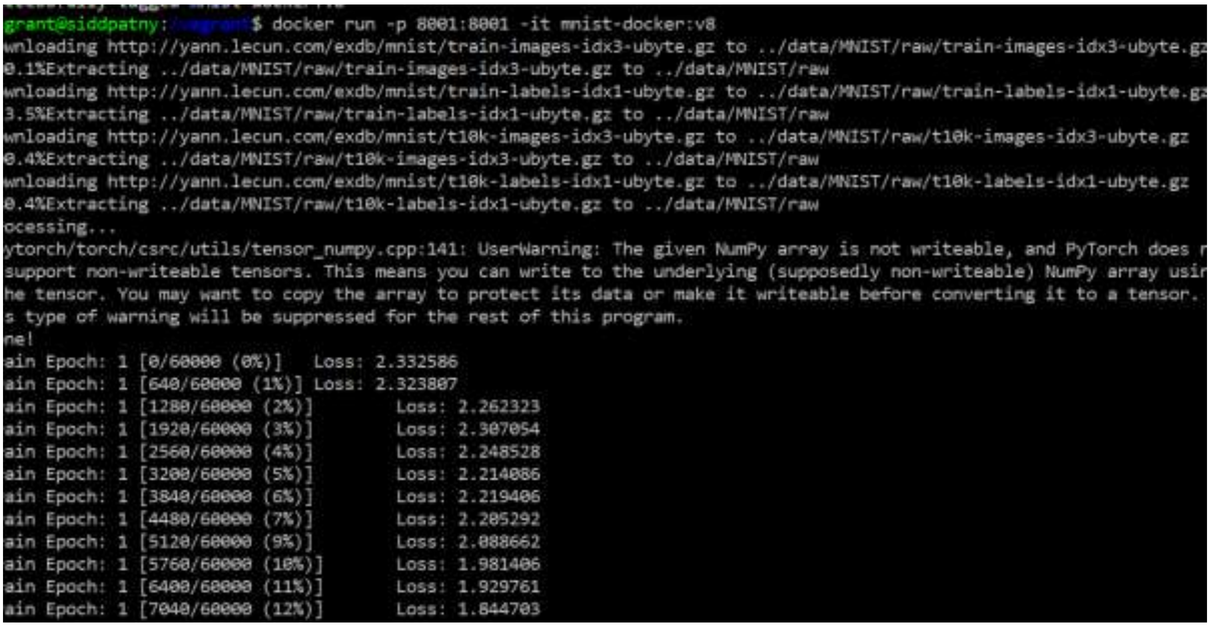
Report:

Docker is popular tool to build images, create containers for different applications or deployments and use volumes to mount and persist user data.

For this assignment, I created a virtualbox VM using vagrant with the vagrantfile provided in this repo [1] as the base. I increased memory using the vagrant plug-in [2]. I also changed the base image for docker to the tiny python 3.6 container. This enabled the image to be smaller than a larger Ubuntu container. The VM ran with the latest 64-bit version of Ubuntu with docker and IBM Cloud CLI installed.

Once the VM was ready. I could ssh into it. I added the code and dockerfile to the VM directory and built the docker image. The MNIST code was cloned from the pytorch-cli repository [3]. The docker container installed the requirements using pip and had the final layer of the code. This allowed for easy upgrade as the final layer of the code changes the most. We build this and give it a new tag (mnist-train) as the code downloaded the MNIST dataset, trained the model and saved the saved the model state in a file (mnist_cnn.pt). Optionally, I also pushed the image to my docker hub using docker login and docker tag (image id) (tag name). Since the image is large (2.3GB) because of the pytorch dependencies (in the second to last layer), this could take some time. The image can be found it siddpatny/mnist-train.

Once the image was built, I could run the container and see the training output on the terminal as shown in the image below. This process of containerizing was convenient and efficient as I could load the VM or a new VM with the container image by specifying the docker image and it would set up the container with the dependencies installed and run the training with a few simple steps. Since the code was currently running on a VM in my local machine. It was much slower than running it on a cloud provider or HPC cluster. However on an old system which I have been using for years with multiple environments, it enabled me to isolate this application in the container and install the dependencies and run the train, with very little configuration and no conflict with other projects.



```
grant@siddpatny:~/vagrant$ docker run -p 8001:8001 -it mnist-docker:v8
wnloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ../data/MNIST/raw/train-images-idx3-ubyte.gz
0.1%Extracting ../data/MNIST/raw/train-images-idx3-ubyte.gz to ../data/MNIST/raw
wnloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ../data/MNIST/raw/train-labels-idx1-ubyte.gz
3.5%Extracting ../data/MNIST/raw/train-labels-idx1-ubyte.gz to ../data/MNIST/raw
wnloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ../data/MNIST/raw/t10k-images-idx3-ubyte.gz
0.4%Extracting ../data/MNIST/raw/t10k-images-idx3-ubyte.gz to ../data/MNIST/raw
wnloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to ../data/MNIST/raw/t10k-labels-idx1-ubyte.gz
0.4%Extracting ../data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ../data/MNIST/raw
ocessing...
ytorch/torch/csrc/utils/tensor_numpy.cpp:141: UserWarning: The given NumPy array is not writeable, and PyTorch does n
support non-writeable tensors. This means you can write to the underlying (supposedly non-writeable) NumPy array usin
he tensor. You may want to copy the array to protect its data or make it writeable before converting it to a tensor.
s type of warning will be suppressed for the rest of this program.
ne!
ain Epoch: 1 [0/60000 (0%)]   Loss: 2.332586
ain Epoch: 1 [640/60000 (1%)] Loss: 2.323807
ain Epoch: 1 [1280/60000 (2%)]        Loss: 2.262323
ain Epoch: 1 [1920/60000 (3%)]        Loss: 2.307054
ain Epoch: 1 [2560/60000 (4%)]        Loss: 2.248528
ain Epoch: 1 [3200/60000 (5%)]        Loss: 2.214086
ain Epoch: 1 [3840/60000 (6%)]        Loss: 2.219406
ain Epoch: 1 [4480/60000 (7%)]        Loss: 2.205292
ain Epoch: 1 [5120/60000 (9%)]        Loss: 2.088662
ain Epoch: 1 [5760/60000 (10%)]       Loss: 1.981406
ain Epoch: 1 [6400/60000 (11%)]       Loss: 1.929761
ain Epoch: 1 [7040/60000 (12%)]       Loss: 1.844703
```

References

1. https://github.com/ihchung/cloudml/tree/master/vg-ibmcloud
2. https://github.com/sprotheroe/vagrant-disksize
3. https://github.com/nyuspring2020/pytorch-cli